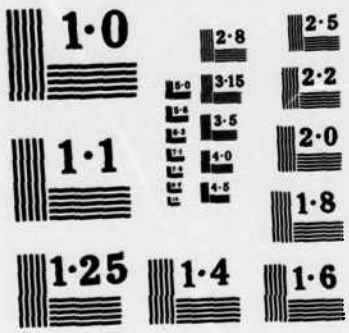
AD-A158 119 ADVANCED AVIONICS COMPUTER ARCHITECTURE VOLUME 1 EXECUTIVE SUMMARY(U) SANDERS ASSOCIATES INC NASHUA NH L GREENSPAN ET AL. MAY 85 AFWAL-TR-85-1041-VOL-1 1/1

UNCLASSIFIED F33615-79-C-1935 F/G 9/2 NL



NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

AFWAL-TR-85-1041

ADVANCED AVIONICS COMPUTER ARCHITECTURE

VOLUME I - EXECUTIVE SUMMARY

LAWRENCE GREENSPAN
RONALD SINGLETARY

SANDERS ASSOCIATES, INC.
95 CANAL STREET
NASHUA, NEW HAMPSHIRE 03061-2034

AD-A158 119

MAY 1985

FINAL REPORT FOR PERIOD MAY 1980 - NOVEMBER 1984

DTIC FILE COPY

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

DTIC
ELECTE
AUG 15 1985
S D
A

85 8 9 081

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Guy A Vince

GUY A. VINCE
Project Engineer, Information Processing
Technology Branch
Avionics Laboratory

Jerry L Covert

JERRY L. COVERT, Acting Chief
Information Processing Technology
Branch
Avionics Laboratory

FOR THE COMMANDER

Raymond D. Bellem

RAYMOND D. BELLEM, COL, USAF
Deputy Chief
System Avionics Division
Avionics Laboratory

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/AAAT-2 *W-PAFB, OH 45433 to help us maintain a current mailing list".*

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

ECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approval for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFWAL-TR-85-1041, Vol I
6a. NAME OF PERFORMING ORGANIZATION SANDERS ASSOCIATES, INC. — 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Avionics Laboratory (AFWAL/AAAT) AF Wright Aeronautical Laboratories (AFSC)
6c. ADDRESS (City, State and ZIP Code) 95 Canal Street Nashua NH 03061-2034	7b. ADDRESS (City, State and ZIP Code) WPAFB OH 45433-6543
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Avionics Laboratory — 8b. OFFICE SYMBOL (If applicable) AFWAL/AAAT	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-79-C-1935
8c. ADDRESS (City, State and ZIP Code) WPAFB OH 45433-6543	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: 62204F — PROJECT NO.: 2003 — TASK NO.: 04 — WORK UNIT NO.: 19
11. TITLE (Include Security Classification) Advanced Avionics Computer Architecture, Vol I	
12. PERSONAL AUTHOR(S) Lawrence Greenspan, Ronald Singletary	
13a. TYPE OF REPORT FINAL — 13b. TIME COVERED FROM 5/80 TO 11/84	14. DATE OF REPORT (Yr., Mo., Day) 1985 May — 15. PAGE COUNT 16
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
High Level Language Ada Machine;
Semantic Gap Reduction;
Language-Directed Architecture; → (over)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This exploratory development program was originally aimed at developing a computer with features to specifically support the JOVIAL (J73) programming language with considerations to Ada. Later, the program was redirected to modify the instruction set architecture (ISA) to more fully support Ada and increase performance.

The new ISA supports most of the standard functions found in most ISA, but gives additional supports in: the Ada package concept, processing arrays and records, unconstrained arrays, dynamic storage allocation, detecting dangling references, detecting undefined variables, Ada-like exception handling, case instructions, for-loop instructions, Ada-like parameter passing, Ada-like tasking instructions and IEEE-standard floating point data types. Keywords; ——

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ OTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Guy Vince	22b. TELEPHONE NUMBER (Include Area Code) 57706 — 22c. OFFICE SYMBOL AFWAL/AAAT-2

11. Title - Executive Summary

18. Non-Von Neumann Architecture;
Object Oriented Architecture;
Capability Based Addressing.

PREFACE

The contents of the document are technically accurate, and no sensitive items, detrimental ideas, or deleterious information are contained therein. Furthermore, the views expressed in the document are those of the author(s) and do not necessarily reflect the views of the Avionics Laboratory, the Air Force Systems Command, the United States Air Force, or the Department of Defense.

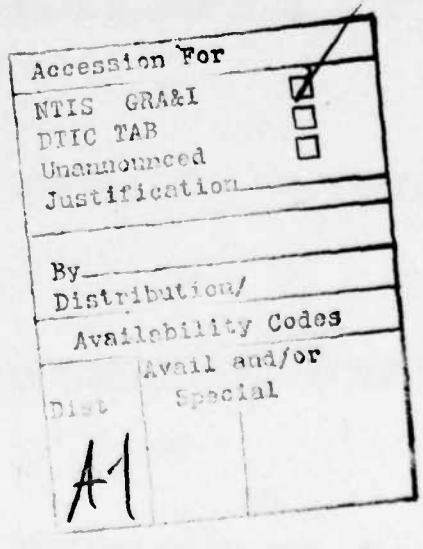


TABLE OF CONTENTS

Page

1. Review of Program.....1
2. Progress.....1
3. Future Work.....3

Appendix

Preliminary Comparison of the HLLM vs. the 1750A....A-1

1. Review of Program

This exploratory development program was originally aimed at developing a computer with features to specifically support the JOVIAL (J73) programming language with consideration given to Ada. Later, the program was redirected to modify the Instruction Set Architecture (ISA) to more fully support Ada and increase performance. This has caused a drastic rewrite of the ISA. Further, a strong emphasis has been placed on improving performance. Thus, variable length data, descriptors, and instructions have been eliminated in favor of fixed length 36-bit words, tags have been reduced in length to 4 bits and do not contain type information, unique names (requiring table look-up) have been removed from all pointers except those that point to data objects when Ada unchecked storage deallocation is programmed; a register file has been added for saving temporaries during arithmetic operations, base-offset addressing of array components, and parameter passing; each activation record now has a stack for expression evaluation; implicit type conversion of arithmetic data types has been dropped in favor of explicit conversion (via instruction); a unique dual data memory system has eliminated the need to physically move data tags and initial values from a data template to an activation record during a subprogram call or task creation; a 4-level pipeline design is planned allowing instruction fetch, address computation, operand retrieval/result storage (in a memory unit that is separate from instruction memory), and execution to proceed in parallel.

2. Progress

Support for the following features is now provided in the ISA:

- the Ada package concept, including packages nested in library (non-nested) packages, task programs, or subprograms.

- processing of arrays and slices including a mode in which the machine computes the component address and checks indexes vs. bounds for all dimensions and a mode in which components are referenced via "base address plus offset" addressing (when the base address of the array has been loaded into a register); logical and assignment operations between whole arrays or slices are also supported.

- unconstrained arrays and records with unconstrained array components (array bounds supplied at run time).
- dynamic storage allocation for the creation of data objects and task objects in support of the evaluation of allocators in Ada.
- detection of dangling references when data objects are explicitly destroyed (possible if the Ada generic library procedure, UNCHECKED DEALLOCATION, is called).
- detection of undefined variables.
- Ada exception handling mechanism, including proper handling of exceptions during task creation, activation, and rendezvous; predefined Ada exceptions and user-defined exceptions are detected and processed.
- zero, one, two, and three-operand instructions with immediate, memory, stack, or register operands.
- initial data values set by compiler without sacrificing properties of recursion/reentrancy of subprograms.
- case and for-loop instructions.
- up-level addressing via display registers.
- Ada in, out, and in-out parameters passed via register or memory-memory transfer.
- creation, activation, scheduling, rendezvous, termination, and dependency rules of tasks.
- Ada context clause (USE/WITH) implemented via pointers to global data areas of external packages (pointers contain access rights).
- predefined language attributes (image, value, upper and lower array bounds, array length, array size, task callable, calling task count).
- 32 and 64-bit IEEE-standard floating point.
- Ada-standard I/O modes (direct, sequential, and text I/O.).

A preliminary comparison was made of the performance of the low level features of the new ISA and the 1750A (see Appendix). The results of the comparison showed that the 1750A required 52% more instruction fetches and 79% more data transfers than the new ISA.

The following sections of the ISA have been written (168 pages total):

Table of Contents (5 pages)
Section 1.4- ISA Summary (4 pages)
Section 2 - Storage Objects (10 pages)
Section 3 - Data Formats (19 pages)
Section 4 - Instruction Formats (13 pages)
Section 6 - Subprograms (18 pages)
Section 7 - Packages (13 pages)
Section 8 - Dynamic Storage Allocation/Deallocation (5 pages)
Section 9 - Tasks (47 pages)
Section 10 - Pointers (11 pages)
Appendix A - Examples of Arrays (15 pages)
Appendix B - Task Dependencies (8 pages)

Except for Section 14 (input-output) which now adheres to Ada more closely than the old ISA (dated 19 March 1982) and a few changes such as no implicit type conversion performed in arithmetic instructions, the instructions not yet described in the new ISA Specification are essentially the same as described in the old ISA Specification. The major differences between the ISAs are reflected in the sections that have been written.

3. Future Work

Although not yet incorporated into the ISA Specification, the following modifications are planned:

(a) more efficient support for Ada character strings.

Strings will be handled as arrays of data type V32, each word containing four characters. Characters in a string will be accessible by the usual methods of addressing a component in an array: (1) via addressing the array header with subscripts and an operand qualifier following in the instruction stream (the operand qualifier selects 1 of 4 characters), (2) via directly addressing the word containing the desired character followed by an operand qualifier that selects the character, (3) via array base register-offset addressing, including full size and compact instruction formats in which the offset "number of bytes".

(b) revised format (FMT) codes to significantly improve performance, especially of looping and array processing via registers.

Fewer (23 vs 28) and more powerful instruction formats are planned. An important change will be to designate a stack operand by "cell offset=0" (similar to designating register operands by cell offsets in the range 1..31); this replaces numerous FMT codes that designate a stack operand in combination with other memory, register, and stack operands. In the following summary of format modes, any operand designated as "memory" can be in memory, on the stack, or in a register, depending on the value of the cell offset:

- 3-operand memory-memory-memory, immediate-memory-memory, immediate-memory-immediate, memory-memory-immediate, and memory-immediate-memory formats (9-bit cell offset and 8-bit immediate value). All memory operands are located in the local environment (address space is implicitly equal to the current nesting depth).

- 2-operand memory-memory, memory-immediate, immediate-memory, and immediate-immediate formats (12-bit cell offset and 12-bit immediate value). Again, memory operands are in the local environment.

- full-size single memory operand format (4-bit address space and 20-bit cell offset).

- full size single immediate operand (20 bits).

- 4-operand register-register-register-immediate format (5-bit register specifiers and 8-bit immediate value). This important mode can be used for complete loop control in which the three registers contain the loop control variable, the increment (decrement) amount, and the limit value; the immediate operand is the label (branch amount relative to the location of the loop-up or loop-down instruction). Another use for this mode is with the new "relational/increment" instructions (refer to item c) in which one register contains an array base address, another register contains the value to be compared with the addressed array component, and the third register contains the increment amount; as before, the immediate operand is the branch amount.

- 2-operand base register-offset-memory and memory-base register-offset formats (9-bit cell offset, 3-bit base register specifier, 8-bit offset value). The memory operand is in the local environment. These modes are used

in instructions that operate on an array component (referenced by base + offset addressing) and a memory operand.

- 2-operand base register-memory-immediate and immediate-base register-memory formats (9-bit cell offset, 3-bit base register specifier, 8-bit immediate value). The memory operand is in the local environment. These modes are used in instructions that operate on an array component (referenced by base + index addressing) and an immediate operand. The index value is the memory operand.

- 2-operand base register-offset-immediate and immediate-base register offset formats (3-bit base register specifier, 8-bit offset and immediate values). This mode is used in instructions that operate on an array component (referenced by base + offset addressing) and an immediate operand.

- single operand base register-offset format (16-bit offset value and 3-bit base register specifier). This mode is used in instructions that operate on an array component (referenced by base + offset addressing).

- 2-operand base register-index-base register-index format (3-bit base register specifiers and 5-bit index register specifiers). This mode is used in instructions that operate on two array components (each referenced by base + index register addressing).

- 2-operand base regiser-index-memory and memory-base register-index formats (3-bit base register specifier fields 5-bit index register specifiers, and 9-bit cell offset). Memory operands are in the local environment. This mode is used in instructions that operate on an array component (referenced by base + index register addressing) and a memory operand.

- single operand base register-memory format (3-bit base register and 12-bit cell offset). The memory operand is in the local environment. This mode is used in instructions that operate on an array component (referenced by base + index addressing). The index value is the memory operand.

Note that array offsets are non-modifiable immediate values whereas indexes are variables in memory or registers.

(c) new group of relational instructions (IF+).

These instructions compare the value of the array component (comparand 1) addressed by an array base register with the value in another register (comparand 2) for equality, non-equality, less than, greater than, less than or equal to, or greater than or equal to. The base address is incremented and a branch is taken if the test is false but the program falls through to the next instruction if the test is true. The 4-operand register-register-register-immediate format can be used with these instructions. Here, the registers designated in the instruction contain the array base address (that is incremented), the increment amount, and the value of comparand 2. If the result of the relational test between the comparands is false, the base register is incremented by the increment amount and a branch is taken to the location which is the sum of the current instruction location and the value of the immediate operand; otherwise (test true), the next instruction is executed.

Based on the very positive preliminary results of performance comparisons between the 1750A and the low level features of the new ISA and the fact that the high level features of the new ISA support the many unique language characteristics of Ada more strongly than any existing military computer, it is recommended that this program be continued. The following tasks need to be done:

- complete the ISA Specification
- thoroughly evaluate the new ISA.

Appendix

Preliminary Comparison of the HLLM vs. the 1750A

The High Level Language Machine (HLLM) has been tailored specifically to the programming language Ada with high level support for packages, activation records, task objects. access types, arrays and records. In addition, conventional low level support for compiler optimizations (i.e. general purpose registers and stacks for evaluating expressions) has been included.

The following analysis compares the low level features of the 1750 with the low level features of the HLLM. As a basis for comparison, four segments of Ada statements are compiled into 1750 instructions and HLLM instructions. The four segments of Ada statements were taken from a document written by Boeing entitled: "Technical Requirements Description for VAX-Hosted Ada/1750A Cross Compiler". For each segment of Ada statements, 1750A code of the expected quality from the phase 2 compiler was presented. The first segment of Ada code in example #1 was extracted from a QUICKSORT program and illustrates scanning of an array for a value that is equal to or greater than some base value. The second segment of Ada statements in example #2 illustrates an iterative search of a two-dimension array which locates the largest absolute value. The third segment of Ada statements in example #3 represents a complex expression requiring index checking, subscript calculations and expression evaluation to determine the value to be assigned. Finally, example #4 contains a segment of Ada statements illustrating an inline function which returns the largest value of two numbers.

Each example includes the segment of Ada statements, 1750 code, HLLM code and a comparison of the number of instruction words and data words transferred to and from memory during execution of the segment of code. A summary of the results of the four examples is shown below.

EXAMPLE#	# of Instruction Wds Fetched 1750A	HLLM	# of Data Wds Transferred 1750A	HLLM
1	3	1	1	1
2	7	4	2	1
3	26	20	20	10
4	5	2	2	2
Total	41	27	25	14

As a whole the 1750A requires 52% more instruction fetches and 79% more data transfers than the HLLM.

EXAMPLE

#1

ADA:

```
                    .
                    .
                    .
loop
 I := I+1;                  -Constraint check suppressed
 exit when A(I)>=V;         -V is a Floating Point Number
end loop;
                    .
                    .
                    .
```

1750A:

```
$0 inc R12,2                -Assume R12 contains A(I-1) address
  fcb R12,0                 -R0,R1 contain the value V
  bit $0
```

HLLM:

```
$0 ifi>+ BR1,R1,R2,$0       -Assume BRI contains A(I) address
                            -R2 contains the increment value
                            -R1 contains the value V
```

EXAMPLE

#2

ADA:

```
                    .
                    .
                    .
S :=0.0;
for J in 1..N
loop
 T := ABS(A(I,J));      -bounds of A are STATIC
 if T>S then S:= T;     -Constraint checks are suppressed
 end if;
end loop;
                    .
                    .
                    .
```

1750A:

```
$0 lb R12,0             -Assume R12 contains A(I-1) address
  fabs R2,R2            -R2 contains the value T
  cr R2,R3              -R3 contains the value S
  bge $1                -R4 contains the remaining count
  lr R3,R2
$1 inc R12,2
  soj R4,$0
```

HLLM:

```
                        -Assume BR1 contains A(I,1)address
$0 absf2 BR1:XR5,R2     -R2 contains the value T
   ifi>R2,R3,$1         -R3 contains the value S
   move R2,R3           -R1 contains size - array component
$1 loopup R1,R5≤R4,$0   -R5 contains loop control value
                        -R4 contains limit value
```

1750 Instruction Wds Fetched = 7 HLLM Instruction Wds Fetched = 4
1750 Data Words Read/Written = 2 HLLM Data Words Read/Written = 1

-The 1750 requires 75% more instruction words fetch than the HLLM.
-The 1750 requires 100% more data words transfer than the HLLM.

EXAMPLE

#3

ADA:

```
                    .
                    .
                    .
                    -Constraint checking is not
                      suppressed
                      A3(I,J,K) := (A3(I,J,K-1) +
                      A3(I,J,K+1) + A3(I,J-1,K) +
                      A3(I,J+1,K) + A3(I-1,J,K) +
                      A3(I+1,J,K))/6.0;
                    .
                    .
                    .
```

1750A:

```
cbl R14,D0          -Assume R12 contains address of A3
bnz $0              -A3 array is static
cbl R6,D0
bnz $0
cbl R8,D0           -R14 contains the value I
bnz $0              -R6 contains the value J
mim R14,2           -R8 contains the value K
mim R6,10
mim R8,40
aim R14,118
ar R14,R6
ar R14,R8
ar R14,R12
dlb R14,-40
ab R14,+40
ab R14,-10
ab R14,+10
ab R14,-2
ab R14,+2
dim R14,6
stb R14,0
$0 equ $
```

EXAMPLE
#3
(continued)

```
        astri R14,C0,C0        -Assume BR1 contains address of A3
        astri R6,C0,C0         -R14 contains the value I
        astri R8,C0,C0         -R6 contains the value J
        muli3 2,R14,Stack      -R8 contains the value K
        muli3 10,R6,Stack      -BR2 will contain address of
        muli3 40,R8,Stack         A3(I,J,K)
        addi2 118,Stack
HLLM:   addi2 Stack,Stack
        addi2 Stack,Stack
        incb3 Stack,BR1,BR2
        addf3 BR2:-40,BR2:40,Stack
        addf3 BR2:-10,BR2:40,Stack
        addf3 BR2:-2,BR:2,Stack
        addf2 Stack,Stack
        addf2 Stack,Stack
        divf3 Stack,6,BR2:0
```

1750 Instruction Wds Fetched =26 HLLM Instruction Wds Fetched =20
1750 Data Words Read/Written =20 HLLM Data Words Read/Written =10

-The 1750 requires 30% more instruction words fetch than the HLLM.
-The 1750 requires 100% more data words transfer than the HLLM.

EXAMPLE

#4

ADA:

```
   function MAX (A, B: INTEGER) return INTEGRER is
   begin
     if A>B then Return A;
            else Return B;

   end if;
end MAX;
pragma INLINE(MAX);

                    .
                    .
                    .
   I := MAX(0,I);
   PROC;
                    .
                    .
                    .
```

1750A:

```
1 R1,I
  bge $0
$0 equ $
```

HLLM:

```
  ifi> 0,C0,$0
  move 0,C0
$0 equ $
```

1750 Instruction Wds Fetched =5 HLLM Instruction Wds Fetched =2
1750 Data Words Read/Written =2 HLLM Data Words Read/Written =2

-The 1750 requires 150% more instructions words fetched than HLLM,
-The data transfer rate is the same for both machines.

END

FILMED

9-85

DTIC

Total	41	27	25	1%

As a whole the 1750A requires 52% more instruction fetches and 79% more data transfers than the HLLM.